DevOps Shack

NEXUS3

Nexus 3 is a popular artifact repository manager used in software development and DevOps workflows. It is designed to store and manage binary artifacts such as JAR files, Docker images, npm packages, and more. Nexus 3 is developed by Sonatype and provides a centralized location for teams to store, retrieve, and manage dependencies and artifacts in a secure and efficient manner. Below, I'll explain the key features and components of Nexus 3 in detail:

1. **Artifact Repository Manager**:
    - Nexus 3 primarily functions as an artifact repository manager, allowing organizations to store and manage various types of binary artifacts. These artifacts can include libraries, dependencies, and build outputs required for software development projects.
2. **Support for Multiple Repository Formats**:
    - Nexus 3 supports a wide range of repository formats, including:
        - **Maven**: For Java-based projects, Maven Central compatibility.
        - **Docker**: Container images and registries.
        - **npm**: Node.js package manager for JavaScript and Node.js projects.
        - **NuGet**: Package manager for .NET applications.
        - **Raw**: Generic storage for any binary format.
        - **Yum**: For RPM package management in Linux environments.
        - **PyPI**: Python package manager repository support.
3. **Proxy and Caching**:
    - Nexus 3 allows you to set up proxy repositories that cache external repositories like Maven Central or Docker Hub. This feature improves build performance by reducing the need to repeatedly download dependencies from external sources.
4. **Hosting Private Repositories**:
    - Nexus 3 enables organizations to host private repositories to store their proprietary or custom artifacts. This is essential for managing internal libraries and ensuring data security.

5. **Search and Indexing**:
    - Nexus 3 provides robust search capabilities, making it easy to find and retrieve artifacts quickly. It indexes repositories, allowing you to search for artifacts by name, version, and other metadata.
6. **Security and Access Control**:
    - Nexus 3 includes fine-grained access control features. You can define roles and permissions to restrict or grant access to specific users or groups, ensuring that sensitive artifacts are protected.
7. **Integration with CI/CD Tools**:
    - Nexus 3 integrates seamlessly with popular CI/CD (Continuous Integration/Continuous Deployment) tools like Jenkins, Travis CI, and CircleCI. This integration ensures that artifacts are automatically published and retrieved during the build and deployment processes.
8. **Lifecycle Management**:
    - Nexus 3 supports the management of artifact lifecycles. You can define retention policies, promote artifacts through different stages (e.g., from development to production), and track the history of changes.
9. **Repository Health and Monitoring**:
    - It provides monitoring and reporting features to track repository health, performance, and usage. You can identify storage issues and optimize your repository manager accordingly.
10. **RESTful API**:
    - Nexus 3 offers a RESTful API that allows you to automate various tasks, such as uploading and downloading artifacts, configuring repositories, and managing permissions.
11. **High Availability and Scalability**:
    - Nexus 3 can be set up in a high-availability configuration for improved reliability. It can scale horizontally to handle increased artifact storage and retrieval demands.
12. **User-Friendly Web Interface**:
    - Nexus 3 includes a web-based user interface that makes it easy to manage repositories, configure settings, and perform administrative tasks.
13. **Plugin Ecosystem**:
    - Nexus 3 has a plugin system that allows you to extend its functionality to meet specific requirements or integrate with other tools and systems.

In summary, Nexus 3 is a powerful artifact repository manager that plays a crucial role in modern software development and DevOps pipelines. It offers features for artifact storage, caching, security, and integration with various development tools, making it an essential component of many software development environments.

NEXUS3 INSTALLATION by Linux commands

sudo apt install openjdk-8-jdk -y

cd /opt

wget https://download.sonatype.com/nexus/3/nexus-3.59.0-01-unix.tar.gz

tar -xvf nexus-3.59.0-01-unix.tar.gz

 # Create user nexus

adduser nexus

chown -R nexus:nexus nexus-3.59.0-01/

chown -R nexus:nexus sonatype-work/

vi nexus-3.59.0-01/bin/nexus.rc

in "" put nexus --->"nexus"

/opt/nexus-3.59.0-01/bin/nexus start

Installation using Docker (Easy Way)

To install Nexus 3 using Docker and retrieve the initial admin password, you can use the following shell commands:

1. Install Nexus 3 Using Docker:

docker run -d -p 8081:8081 --name nexus sonatype/nexus3

This command pulls the Nexus 3 Docker image from the official Sonatype repository and runs it as a detached container (-d). It exposes the Nexus web interface on host port 8081 (-p 8081:8081) and names the container "nexus."

2. Retrieve the Initial Admin Password:

Wait for Nexus to start, and then retrieve the initial admin password. You can do this by checking the logs or using the following command:

docker ps

note down container_ID

docker exec -it container_ID /bin/bash

cat sonatype-work/nexus3/admin.password

This command uses docker exec to execute a command inside the running "nexus" container. The command cat /nexus-data/admin.password prints the initial admin password to the terminal.

3. Access Nexus 3 Web Interface:

Open a web browser and go to http://localhost:8081. Log in with the username "admin" and the password retrieved in the previous step.

Note: Make sure to wait for Nexus 3 to fully initialize before attempting to retrieve the admin password.

Cleanup (Optional):

If you want to stop and remove the Nexus 3 Docker container after testing, you can use the following commands:

```
docker stop nexus
```

```
docker rm nexus
```

These commands stop and remove the "nexus" container.

Remember to adjust the Docker commands based on your specific requirements and environment. Additionally, ensure that Docker is installed and configured on your system before running these commands.

ADD in POM

```
<project>
  <!-- ... other project information ... -->

  <distributionManagement>
    <repository>
      <id>maven-releases</id>
      <url>NEXUS-URL/repository/maven-releases/</url>
    </repository>
    <snapshotRepository>
      <id>maven-snapshots</id>
      <url>NEXUS-URL/repository/maven-snapshots/</url>
    </snapshotRepository>
  </distributionManagement>

  <!-- ... other project configuration ... -->
</project>
```

PIPELINE

```
pipeline {
    agent any
    tools{
        jdk 'jdk17'
        maven 'maven3'
    }
    environment{
        SCANNER_HOME= tool 'sonar-scanner'
    }

    stages {
        stage('git-checkout') {
            steps {
                git 'https://github.com/jaiswaladi2468/BoardgameListingWebApp.git'
            }
        }

        stage('Code-Compile') {
            steps {
                sh "mvn clean compile"
            }
        }

        stage('Unit-Test') {
            steps {
                sh "mvn clean test"
            }
        }

        stage('Trivy Scan') {
            steps {
                sh "trivy fs ."
            }
        }

        stage('OWASP Dependency Check') {
            steps {
                dependencyCheck additionalArguments: ' --scan ./ ', odcInstallation: 'DC'
                    dependencyCheckPublisher pattern: '**/dependency-check-report.xml'
            }
        }

        stage('Sonar Analysis') {
            steps {
                withSonarQubeEnv('sonar'){
                    sh ''' $SCANNER_HOME/bin/sonar-scanner -Dsonar.projectName=BoardGame \
                    -Dsonar.java.binaries=. \
                    -Dsonar.projectKey=BoardGame '''
                }
            }
```

```
        }

        stage('Code-Build') {
            steps {
              sh "mvn clean package"
            }
        }

        stage('Deploy To Nexus') {
            steps {
              withMaven(globalMavenSettingsConfig: 'e7838703-298a-44a7-b080-a9ac14fa0a5e') {
                  sh "mvn deploy"
              }
            }
        }
    }
}
```

Pipeline for downloading the Artifact

```
pipeline {
    agent any
    tools{
        jdk 'jdk17'
        maven 'maven3'
    }
    environment{
        SCANNER_HOME= tool 'sonar-scanner'
    }

    stages {
        stage('git-checkout') {
            steps {
                git 'https://github.com/jaiswaladi2468/BoardgameListingWebApp.git'
            }
        }

        stage('Code-Compile') {
            steps {
                sh "mvn clean compile"
            }
        }

        stage('Unit-Test') {
            steps {
                sh "mvn clean test"
            }
        }

        stage('Trivy Scan') {
            steps {
                sh "trivy fs ."
            }
        }

        stage('OWASP Dependency Check') {
            steps {
                dependencyCheck additionalArguments: ' --scan ./ ', odcInstallation: 'DC'
                    dependencyCheckPublisher pattern: '**/dependency-check-report.xml'
            }
        }

        stage('Sonar Analysis') {
            steps {
                withSonarQubeEnv('sonar'){
                    sh ''' $SCANNER_HOME/bin/sonar-scanner -Dsonar.projectName=BoardGame \
                    -Dsonar.java.binaries=. \
                    -Dsonar.projectKey=BoardGame '''
                }
```

```
        }
    }

    stage('Download JAR with Credentials') {
        steps {
            script {
                withCredentials([usernamePassword(credentialsId: 'your-credentials-id',
usernameVariable: 'user', passwordVariable: 'pass')]) {
                    def jarUrl = 'https://example.com/path/to/your.jar'

                    sh "curl -u $user:$pass -O $jarUrl"
                }
            }
        }
    }

    stage('Build & Deploy To Nexus') {
        steps {
            withMaven(globalMavenSettingsConfig: 'e7838703-298a-44a7-b080-a9ac14fa0a5e') {
                sh "mvn deploy"
            }
        }
    }
  }
}
```

Publishing Node.js Artifacts to Nexus using Jenkins

Step 1: Create a Custom .npmrc File in Jenkins

1. **Navigate to Jenkins Configuration**:
    - Go to Manage Jenkins > Managed Files.
2. **Create a New Custom File**:
    - Click on Add a new Config.
    - Select Custom file and name it .npmrc.
3. **Add Authentication Details**:
    - Convert your Nexus credentials to base64:

echo -n 'admin:aditya' | base64

    - The output will be: YWRtaW46YWRpdHlh
    - Add the following lines to the .npmrc file:

```
registry=http://13.235.245.200:8081/repository/npm-private

//IP:8081/repository/npm-private/:_auth=YWRtaW46YWRpdHlh
```

Step 2: Jenkins Pipeline Configuration

Create a Jenkins pipeline with the following stages:

```
pipeline {
  agent any

  stages {
    stage('Git') {
      steps {
        git branch: 'main', url: 'https://github.com/jaiswaladi246/NodejS-JEST.git'
      }
    }

    stage('NPM Dependencies') {
      steps {
        nodejs('node20') {
          sh "npm install"
        }
      }
    }

    stage('Publish to Nexus') {
      steps {
        configFileProvider([configFile(fileId: 'npmrc', targetLocation: '.')]) {
          nodejs('node20') {
```

```
                sh "npm publish"
            }
        }
    }
  }
 }
}
```

Explanation:

1. **Git Stage**:
    - The code is checked out from the main branch of the specified GitHub repository.
2. **NPM Dependencies Stage**:
    - The Node.js environment node20 is specified.
    - Dependencies are installed using npm install.
3. **Publish to Nexus Stage**:
    - The .npmrc file created earlier is provided to the pipeline using configFileProvider.
    - The artifacts are published to Nexus using npm publish.

Additional Notes:

- Ensure that the node20 tool is configured in Jenkins under Manage Jenkins > Global Tool Configuration.
- The repourl in the .npmrc file should be replaced with the actual Nexus repository URL.